CS 448

PSO Week -6
Indexing in SimpleDB

SimpleDB indexing

Index- Sequentially ordered file

B+Tree Index

Hash Index

Index Interface

```java
/**
 * This interface contains methods to traverse an index.
 * @author Edward Sciore
 *
 */
public interface Index {

   /**
    * Positions the index before the first record
    * having the specified search key.
    * @param searchkey the search key value.
    */
   public void    beforeFirst(Constant searchkey);

   /**
    * Moves the index to the next record having the
    * search key specified in the beforeFirst method.
    * Returns false if there are no more such index records.
    * @return false if no other index records have the search key.
    */
   public boolean next();

   /**
    * Returns the dataRID value stored in the current index record.
    * @return the dataRID stored in the current index record.
    */
   public RID     getDataRid();

   /**
    * Inserts an index record having the specified
    * dataval and dataRID values.
    * @param dataval the dataval in the new index record.
    * @param datarid the dataRID in the new index record.
    */
   public void    insert(Constant dataval, RID datarid);

   /**
    * Deletes the index record having the specified
    * dataval and dataRID values.
    * @param dataval the dataval of the deleted index record
    * @param datarid the dataRID of the deleted index record
    */
   public void    delete(Constant dataval, RID datarid);

   /**
    * Closes the index.
    */
   public void    close();
}
```

Each record is of the form (dataval,datarid) where dataval is the search key and datarid is the pointer

A RID consists of the block number in the file, and the location of the record in that block.

Example 1: Index Retrieval

```
package simpledb.index;

import java.util.Map;

public class IndexRetrievalTest {
   public static void main(String[] args) {
      SimpleDB db = new SimpleDB("studentdb");
      Transaction tx = db.newTx();
      MetadataMgr mdm = db.mdMgr();

      // Open a scan on the data table.
      Plan studentplan = new TablePlan(tx, "student", mdm);
      UpdateScan studentscan = (UpdateScan) studentplan.open();

      // Open the index on MajorId.
      Map<String,IndexInfo> indexes = mdm.getIndexInfo("student", tx);
      IndexInfo ii = indexes.get("majorid");
      Index idx = ii.open();

      // Retrieve all index records having a dataval of 20.
      idx.beforeFirst(new Constant(20));
      while (idx.next()) {
         // Use the datarid to go to the corresponding STUDENT record.
         RID datarid = idx.getDataRid();
         studentscan.moveToRid(datarid);
         System.out.println(studentscan.getString("sname"));
      }

      // Close the index and the data table.
      idx.close();
      studentscan.close();
      tx.commit();
   }
}
```

Open the Index for “student” table with “MajorId” being the search key for index records

Move to the index record having MajorId or dataval=20, get the RID and go to the corresponding Student record.

Example 2: Index Update

```
// Task 1: insert a new STUDENT record for Sam
//    First, insert the record into STUDENT.
studentscan.insert();
studentscan.setInt("sid", 11);
studentscan.setString("sname", "sam");
studentscan.setInt("gradyear", 2023);
studentscan.setInt("majorid",  30);

//    Then insert a record into each of the indexes.
RID datarid = studentscan.getRid();
for (String fldname : indexes.keySet()) {
   Constant dataval = studentscan.getVal(fldname);
   Index idx = indexes.get(fldname);
   idx.insert(dataval, datarid);
}

// Task 2: find and delete Joe's record
studentscan.beforeFirst();
while (studentscan.next()) {
   if (studentscan.getString("sname").equals("joe")) {

      // First, delete the index records for Joe.
      RID joeRid = studentscan.getRid();
      for (String fldname : indexes.keySet()) {
         Constant dataval = studentscan.getVal(fldname);
         Index idx = indexes.get(fldname);
         idx.delete(dataval, joeRid);
      }

      // Then delete Joe's record in STUDENT.
      studentscan.delete();
      break;
   }
}

// Print the records to verify the updates.
studentscan.beforeFirst();
while (studentscan.next()) {
   System.out.println(studentscan.getString("sname") + " " + studentscan.getInt("sid"));
}
studentscan.close();

for (Index idx : indexes.values())
   idx.close();
```

1. Insert into the “Student” Record
2. Insert the record into each of the Indexes
1. Delete record from Indexes
2. Delete from the “Student” record

B+ Tree

Index

```java
/**
 * Traverse the directory to find the leaf block corresponding
 * to the specified search key.
 * The method then opens a page for that leaf block, and
 * positions the page before the first record (if any)
 * having that search key.
 * The leaf page is kept open, for use by the methods next
 * and getDataRid.
 * @see simpledb.index.Index#beforeFirst(simpledb.query.Constant)
 */
public void beforeFirst(Constant searchkey) {
   close();
   BTreeDir root = new BTreeDir(tx, rootblk, dirLayout);
   int blknum = root.search(searchkey);
   root.close();
   BlockId leafblk = new BlockId(leaftbl, blknum);
   leaf = new BTreeLeaf(tx, leafblk, leafLayout, searchkey);
}

/**
 * Move to the next leaf record having the
 * previously-specified search key.
 * Returns false if there are no more such leaf records.
 * @see simpledb.index.Index#next()
 */
public boolean next() {
   return leaf.next();
}

/**
 * Return the dataRID value from the current leaf record.
 * @see simpledb.index.Index#getDataRid()
 */
public RID getDataRid() {
   return leaf.getDataRid();
}
```

Search Mechanism

```
/**
 * Returns the block number of the B-tree leaf block
 * that contains the specified search key.
 * @param searchkey the search key value
 * @return the block number of the leaf block containing that search key
 */
public int search(Constant searchkey) {
   BlockId childblk = findChildBlock(searchkey);
   while (contents.getFlag() > 0) {
      contents.close();
      contents = new BTPage(tx, childblk, layout);
      childblk = findChildBlock(searchkey);
   }
   return childblk.number();
}
```

```
private BlockId findChildBlock(Constant searchkey) {
   int slot = contents.findSlotBefore(searchkey);
   if (contents.getDataVal(slot+1).equals(searchkey))
      slot++;
   int blknum = contents.getChildNum(slot);
   return new BlockId(filename, blknum);
}
```

B+ Tree

Page

```
/**
 * Open a node for the specified B-tree block.
 * @param currentblk a reference to the B-tree block
 * @param layout the metadata for the particular B-tree file
 * @param tx the calling transaction
 */
public BTPage(Transaction tx, BlockId currentblk, Layout layout) {
   this.tx = tx;
   this.currentblk = currentblk;
   this.layout = layout;
   tx.pin(currentblk);
}
```

```
/**
 * Calculate the position where the first record having
 * the specified search key should be, then returns
 * the position before it.
 * @param searchkey the search key
 * @return the position before where the search key goes
 */
public int findSlotBefore(Constant searchkey) {
   int slot = 0;
   while (slot < getNumRecs() && getDataVal(slot).compareTo(searchkey) < 0)
      slot++;
   return slot-1;
}
```

```
 * Close the page by unpinning its buffer.
 */
public void close() {
   if (currentblk != null)
      tx.unpin(currentblk);
   currentblk = null;
}
```

B+ Tree

Index cont.

```java
/**
 * Insert the specified record into the index.
 * The method first traverses the directory to find
 * the appropriate leaf page; then it inserts
 * the record into the leaf.
 * If the insertion causes the leaf to split, then
 * the method calls insert on the root,
 * passing it the directory entry of the new leaf page.
 * If the root node splits, then makeNewRoot is called.
 * @see simpledb.index.Index#insert(simpledb.query.Constant, simpledb.record.RID)
 */
public void insert(Constant dataval, RID datarid) {
   beforeFirst(dataval);
   DirEntry e = leaf.insert(datarid);
   leaf.close();
   if (e == null)
      return;
   BTreeDir root = new BTreeDir(tx, rootblk, dirLayout);
   DirEntry e2 = root.insert(e);
   if (e2 != null)
      root.makeNewRoot(e2);
   root.close();
}

/**
 * Delete the specified index record.
 * The method first traverses the directory to find
 * the leaf page containing that record; then it
 * deletes the record from the page.
 * @see simpledb.index.Index#delete(simpledb.query.Constant, simpledb.record.RID)
 */
public void delete(Constant dataval, RID datarid) {
   beforeFirst(dataval);
   leaf.delete(datarid);
   leaf.close();
}
```

Thank You